

Propositional Resolution

Second Part: Algorithms

Benjamin Wack

Université Grenoble Alpes

January 2022

Proof by resolution of our running example

- ▶ (H1) : $p \Rightarrow \neg j \equiv \neg p \vee \neg j$
- ▶ (H2) : $\neg p \Rightarrow j \equiv p \vee j$
- ▶ (H3) : $j \Rightarrow m \equiv \neg j \vee m$
- ▶ (\neg C) : $\neg m \wedge \neg p$

Clauses: $\{\neg p \vee \neg j, p \vee j, \neg j \vee m, \neg m, \neg p\}$

$$\frac{\frac{\frac{p \vee j \quad \neg j \vee m}{p \vee m} \quad \neg m}{p} \quad \neg p}{\perp}$$

OR

$$\frac{\frac{\frac{p \vee j \quad \neg p}{j} \quad \neg j \vee m}{m} \quad \neg m}{\perp}$$

OR...

Last course

- ▶ Boolean Algebra
- ▶ Boolean functions
- ▶ Resolution

(1) $A \vdash B$

B is *deduced* from A : there is a proof by resolution of B starting from A .

(2) $A \models B$

B is a *consequence* of A : every model of A is also a model of B .

Today: Correctness

(1) \Rightarrow (2)

Today: Completeness

(2) \Rightarrow (1)

Overview

Correctness

Completeness

Introduction to resolution algorithms

The Davis-Putnam-Logemann-Loveland (DPLL) Algorithm

Complete strategy

Conclusion

Definition

The **correctness** of a deductive system states that all proofs obtained in this system “prove only true statements”.

Correctness of the resolution rule

Theorem 2.1.15

If C is a resolvent of A and B then $A, B \models C$.

Proof.

If C is a resolvent of A and B , then there is a literal L such that $L \in A, L^c \in B$, and $C = (A - \{L\}) \cup (B - \{L^c\})$.

Let ν be an assignment such that $[A]_\nu = 1$ and $[B]_\nu = 1$: let us show that $[C]_\nu = 1$.

- ▶ Suppose that $[L]_\nu = 1$. Therefore $[L^c]_\nu = 0$.
Since $[B]_\nu = 1$, ν is a model of a literal of $(B - \{L^c\})$. Hence $[C]_\nu = 1$.
- ▶ Suppose that $[L^c]_\nu = 1$. Therefore $[L]_\nu = 0$.
Since $[A]_\nu = 1$, ν is a model of $(A - \{L\})$. Hence $[C]_\nu = 1$.

Since every truth assignment is either model of L or L^c , ν is a model of C .

□

Correctness of deduction

Theorem 2.1.16

Let Γ be a set of clauses and C a clause. If $\Gamma \vdash C$ then $\Gamma \models C$.

Proof.

Suppose that there is a proof P of C starting from Γ .

Suppose that for any proof of $\Gamma \vdash D$ **shorter** than P , we have $\Gamma \models D$.

Let us show that $\Gamma \models C$. There are two possible cases:

1. C is a member of Γ , in this case $\Gamma \models C$.
2. $\Gamma \vdash A$ and $\Gamma \vdash B$ (with a shorter proof) and

$$\frac{A \quad B}{C}$$

By induction hypothesis: $\Gamma \models A$ and $\Gamma \models B$.

By correctness of the resolution rule: $A, B \models C$. Hence $\Gamma \models C$.

□

Definition

Completeness for the refutation is the property: If $\Gamma \models \perp$ then $\Gamma \vdash \perp$.

We prove this result for a finite Γ .

$$\Gamma[L := 1]$$

Definition 2.1.18

Let Γ be a set of clauses and L a literal.

$\Gamma[L := 1]$ is obtained by:

- ▶ deleting the clauses containing L
- ▶ removing L^c from the other clauses.

$\Gamma[L := 0]$ is similarly defined by switching the roles of L and L^c .

Remark: the number of variables in Γ has been decreased.

Examples

Example 2.1.19

Let Γ be the set of clauses $\bar{p} + q$, $\bar{q} + r$, $p + q$, $p + r$. We have:

▶ $\Gamma[p := 1] =$

$$\{q, \bar{q} + r\}.$$

▶ $\Gamma[p := 0] =$

$$\{\bar{q} + r, q, r\}.$$

Notice that:

▶ $(\bar{1} + q)(\bar{q} + r)(1 + q)(1 + r) \equiv$

$$q(\bar{q} + r)$$

▶ $(\bar{0} + q)(\bar{q} + r)(0 + q)(0 + r) \equiv$

$$(\bar{q} + r)qr$$

Property of $\Gamma[L := \dots]$

Property 2.1.21

Γ has a model if and only if $\Gamma[L := 1]$ or $\Gamma[L := 0]$ has a model.

Proof.

- \Rightarrow If v is a model of Γ then v is a model of either $\Gamma[L := 0]$ (if $[L]_v = 0$) or $\Gamma[L := 1]$ (if $[L]_v = 1$)
- \Leftarrow If v is a model of $\Gamma[L := i]$ then we can build a model of Γ (by taking $[L]_v = i$)

□

Lemma 2.1.22

Lemma 2.1.22

Let Γ a set of clauses, C a clause and L a literal.

If $\Gamma[L := 1] \vdash C$ then $\Gamma \vdash C$ or $\Gamma \vdash C + L^c$.

Proof.

Idea: we put back L^c in the clauses where it was removed.

- ▶ If $C \in \Gamma[L := 1]$:
 - ▶ either C was in Γ , thus $\Gamma \vdash C$
 - ▶ or C was obtained by removing a L^c , thus $\Gamma \vdash C + L^c$
- ▶ If C is a resolvent of A and B :
 - ▶ either $\Gamma \vdash A$ and $\Gamma \vdash B$, hence $\Gamma \vdash C$
 - ▶ or L^c has to be put back into A or B , thus into C too

□

Completeness of propositional resolution

Theorem 2.1.24

Let Γ be a finite set of clauses. If Γ is unsatisfiable then $\Gamma \vdash \perp$.

Proof

By induction on the **number of variables** in Γ .

- ▶ Base case: Γ has no variable, so $\Gamma = \emptyset$ (impossible, it's valid) or $\Gamma = \{\perp\}$.
- ▶ Inductive step: either we prove directly that $\Gamma \vdash \perp$, or that $\Gamma \vdash x$ and $\Gamma \vdash \bar{x}$.

Corollary 2.1.25

Γ is unsatisfiable if and only if $\Gamma \vdash \perp$.

Presentation of two algorithms

How to “systematically” decide whether Γ is inconsistent or not?

- ▶ **The Davis-Putnam-Logemann-Loveland Algorithm**
“Intelligent” traversal of the possible assignments of Γ
- ▶ **Complete strategy**
Construction of ALL the deductible clauses (resolvents) from Γ

Remark

Exponential solutions in time in the worst case.

Exponential complexity

Remember that two clauses having the same set of literals are equal.

If Γ uses n , then we have at most 2^n distinct clauses deduced from Γ .

Reduction of a set of clauses

In order to accelerate the algorithm, we **reduce** the set of clauses.

How to proceed with reduction?

Remove the **valid** clauses and the clauses **containing another** clause of the set.

Example 2.1.27

The reduction of the set of clauses $\{p + q + \bar{p}, p + r, p + r + \bar{s}, r + q\}$ gives the reduced set:

$$\{\cancel{p + q + \bar{p}}, p + r, \cancel{p + r + \bar{s}}, r + q\}.$$

Justification

Property 2.1.28

A set of clauses E is equivalent to the reduced set of clauses obtained from E .

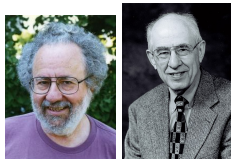
Proof.

- ▶ Removing valid clauses: $x.1 \equiv x$
- ▶ Removing a clause including another clause: $x(x + y) \equiv x$



History

- ▶ Martin Davis (1928-), american mathematician
- ▶ Hilary Putnam (1926-2016), american philosopher, mathematician and computer scientist
- ▶ **resolution** rule (exhaustively used in the first algorithm)
- ▶ Algorithm for satisfiability of boolean formulas (1960)
 - ▶ finds (if possible) **a model of a set of clauses**
 - ▶ initially devised to study first-order formulas
 - ▶ refined in 1962 by M. Davis, G. Logemann and D. Loveland with a branching mechanism
 - ▶ Basis for efficient SAT-solvers
- ▶ Proof of undecidability of Diophantine equations (with Y. Matiyasevich and J. Robinson)



Principle I

Two types of formulae transformations:

1. **preserving the truth value:**
 - ▶ reduction
2. **preserving only satisfiability:**
 - ▶ pure literal elimination
 - ▶ unit resolution

DPLL is (usually) efficient because it uses these two kinds transformations.

Principle II

“Branching/Backtracking” (splitting rule)

- ▶ **Branching**: After simplification, assign to **true** a heuristically chosen variable (branching literal).
- ▶ Continue the algorithm recursively.
- ▶ **Backtracking**: If we arrive to a contradiction, we return to the last choice, and we “branch” by assigning **false** to the chosen variable.

The DPLL Algorithm (figure 2.1)

bool function Algo_DPLL(Γ : set of clauses)

0 Remove the valid clauses from Γ .

If $\Gamma = \emptyset$, return (true).

Else return (DPLL(Γ))

bool function DPLL(Γ : set of non-valid clauses)

The function returns true if and only if Γ is satisfiable.

1 If $\perp \in \Gamma$, return(false).

If $\Gamma = \emptyset$, return (true).

2 Reduce Γ .

3 Remove from Γ the clauses containing a pure literal.

If the set Γ has been modified, goto 1.

4 Apply unit resolution to Γ .

If the set Γ has been modified, goto 1.

5 Pick an arbitrary variable x in Γ

return (DPLL($\Gamma[x := 0]$) or else DPLL($\Gamma[x := 1]$))

Removal of clauses containing a pure literal

Definition 2.3.1

A literal L is **pure** if none of the clauses in Γ contains L^c .

Lemma 2.3.2

Removing clauses with a pure literal preserves satisfiability.

Proof: see exercise 49.

Intuition: assigning $[L]_v$ to 1 is always possible for a pure literal.

Example 2.3.3

Let Γ be the set of clauses

$$(1) \ p + q + r$$

$$(2) \ \bar{q} + \bar{r}$$

$$(3) \ q + s$$

$$(4) \ \bar{s} + t$$

Simplify Γ by removing clauses containing pure literals.

The literals p and t are pure.

Therefore we obtain

$$(2) \ \bar{q} + \bar{r}$$

$$(3) \ q + s$$

The literals \bar{r} and s are pure.

We obtain the empty set.

Therefore Γ has a model (for instance $p = 1, t = 1, r = 0, s = 1$).

Unit resolution

Definition 2.3.4

A **unit clause** is a clause which contains only one literal.

Lemma 2.3.5

Let L be the literal from a unit clause of Γ .

Let Θ be the set of clauses obtained by:

- removing the clauses containing L
- removing L^c **inside** the remaining clauses
- ▶ if Γ contains two complementary unit clauses, then $\Theta = \{\perp\}$.

We apply this process for every unit clause.

Γ has a model if and only if Θ has a model.

Proof: The proof is requested in exercise 50.

Example 2.3.6 Unit resolution

Simplify the following sets of clauses by unit resolution:

► $\Gamma = p + q, \bar{p}, \bar{q}$

q, \bar{q} by unit resolution on \bar{p} , then \perp by UR on \bar{q}
Hence Γ has no model.

► $\Gamma = a + b + \bar{d}, \bar{a} + c + \bar{d}, \bar{b}, d, \bar{c}$

1. a, \bar{a} .
2. \perp

hence Γ has no model.

► $\Gamma = p, q, p+r, \bar{p}+r, q+\bar{r}, \bar{q}+s$

By unit resolution, we obtain: r, s .
This set of clauses has a model, hence Γ has a model.

Removal of valid clauses

Lemma 2.3.7

Let Θ be the set of clauses obtained by removing the valid clauses of Γ .
 Γ has a model iff Θ has a model.

Proof.

\Rightarrow Every model of Γ is clearly a model of Θ , since $\Theta \subseteq \Gamma$.

\Leftarrow Suppose that Θ has a model v .

Let v' be the truth assignment built from v by assigning any value to the variables appearing in Γ but not in Θ .

Every clause C in Γ is:

- ▶ either a clause of Θ , then $[C]_{v'} = [C]_v = 1$
- ▶ or a valid clause, so obviously v' is a model of C .

Hence v' is a model of Γ .

□

The DPLL Algorithm (figure 2.1)

bool function Algo_DPLL(Γ : set of clauses)

0 Remove the valid clauses from Γ .

If $\Gamma = \emptyset$, return (true).

Else return (DPLL(Γ))

bool function DPLL(Γ : set of non-valid clauses)

The function returns true if and only if Γ is satisfiable.

1 If $\perp \in \Gamma$, return(false).

If $\Gamma = \emptyset$, return (true).

2 Reduce Γ .

3 Remove from Γ the clauses containing a pure literal.

If the set Γ has been modified, goto 1.

4 Apply unit resolution to Γ .

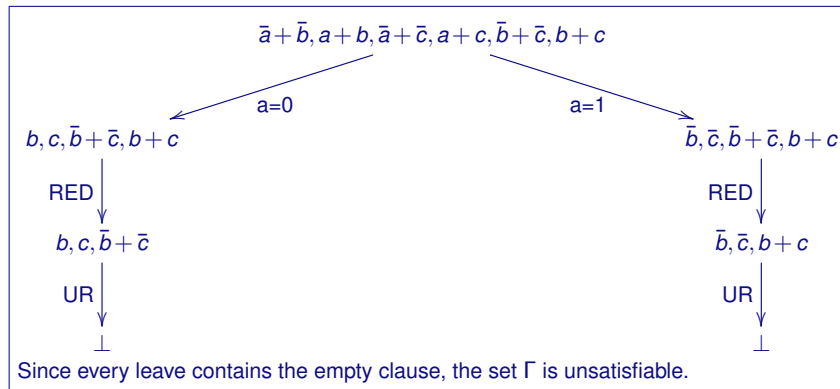
If the set Γ has been modified, goto 1.

5 Pick an arbitrary variable x in Γ

return (DPLL($\Gamma[x := 0]$) or else DPLL($\Gamma[x := 1]$))

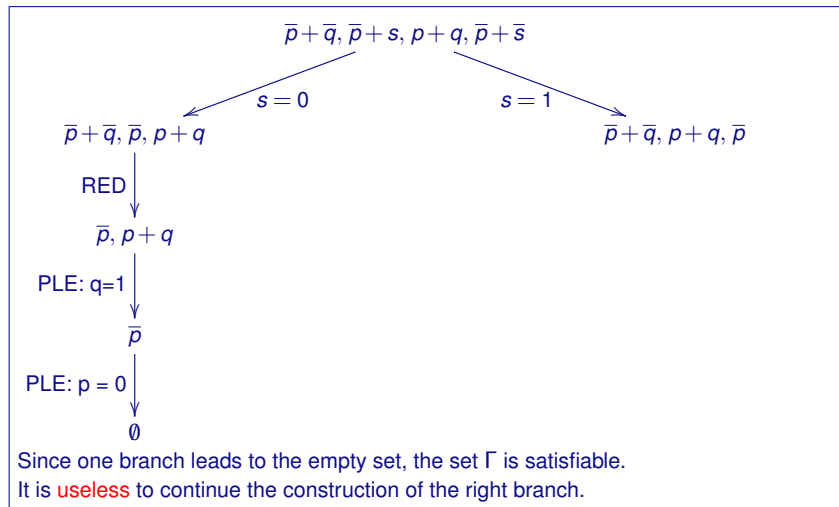
Example 2.3.8

Let Γ be the set of clauses: $\bar{a} + \bar{b}$, $a + b$, $\bar{a} + \bar{c}$, $a + c$, $\bar{b} + \bar{c}$, $b + c$.



Example 2.3.8

Let Γ be the set of clauses: $\bar{p} + \bar{q}$, $\bar{p} + s$, $p + q$, $\bar{p} + \bar{s}$.



Theorems 2.3.9 et 2.3.10

The algorithm Algo_DPLL is correct and terminates.

Termination proof

- ▶ Valid clause removal is only executed once
- ▶ Simplification iteration: the number of clauses strictly decreases
- ▶ Recursive calls: the number of variables strictly decreases

Hence the termination.

Correctness proof

- ▶ Invariant for the simplification loop:

the current value of Γ has a model iff Γ has a model.

see lemma for each simplification.

- ▶ Correctness of recursive calls:

Reminder of property 2.1.21:

Γ has a model iff $\Gamma[x := 0]$ or $\Gamma[x := 1]$ is satisfiable.

So if the recursive calls are correct, the current call is too.

Since the algorithm is correct for a set Γ with no literal, it is correct for any set Γ of clauses.

Remarks 2.3.11 and 2.3.12

- ▶ **Forgetting simplifications:** DPLL is still correct if we forget (once or more) reduction (2), pure literal elimination (3) and/or unit reduction (4).
- ▶ **Choice of the variable (branching literal):**
 - ▶ A good choice for variable x in step (5) is the variable that appears most often.
 - ▶ A better choice is the variable which will lead to the maximum number of simplifications

Cf. Sub-section 2.3.5, for the main branching heuristics

SAT Solveur demo

Problem



- ▶ Each square may either contain a token or not.
- ▶ Two neighbouring squares can never both contain a token.
- ▶ At least two squares must contain a token.

Input of the problem: the length n of the grid

Boolean modelization

- ▶ Each square is associated to a boolean variable (true if the square contains a token)
- ▶ For the Dimacs format, we number the squares 1 to n

Principle of the algorithm: Build all the clauses deduced from Γ

Following the height of the proof trees.

Algorithm

For any integer i

While it is possible to construct new clauses

Build the reduced set of all the clauses having a proof tree of height at most i .

In practice:

Maintain two sequences of the sets of clauses, $\Delta_{i(i \geq 0)}$ and $\Theta_{i(i \geq 0)}$

Result of the algorithm: minimum deduction clauses

Definition 2.1.29

A minimum clause for the deduction from Γ is :

- ▶ a non-valid clause
- ▶ deduced from Γ
- ▶ and containing no other clause deduced from Γ .

Example 2.1.30

$$\Gamma = \{a + \bar{b}, b + c + d\}$$

- ▶ The clause $a + c + d$ is a minimum clause for deduction.
- ▶ But if we add $\bar{a} + c$ to Γ , then $a + c + d$ is not minimal anymore (since we can now deduce $c + d$).

Property

Property 2.1.31

Let Θ be the set of minimum deduction clauses for the set Γ .

Γ is unsatisfiable if and only if $\perp \in \Theta$.

Proof.

- ▶ Suppose $\perp \in \Theta$, then $\Gamma \vdash \perp$, hence by **resolution correctness**, Γ is unsatisfiable.
- ▶ Suppose Γ is unsatisfiable, **by resolution completeness**, $\Gamma \vdash \perp$. Consequently \perp is a minimum clause for deduction from Γ , therefore $\perp \in \Theta$.

□

Two sequences of sets of clauses

Δ_i are the **new** useful clauses

Clauses deduced from Γ by a proof of height i , after removal of:

- ▶ valid clauses
- ▶ clauses including another clause whose proof has height $< i$.

Δ_0 is obtained by reducing Γ .

Θ_i are the **old** clauses still useful

Clauses deduced from Γ by a proof of height $< i$ after removal of:

- ▶ valid clauses
- ▶ clauses including another clause whose proof has height $\leq i$.

Θ_0 is the empty set.

Construction of the sequences $\Delta_{i(i \geq 0)}$ and $\Theta_{i(i \geq 0)}$

Δ_{i+1}

- ▶ Compute all the resolvents of Δ_j and $\Delta_j \cup \Theta_j$
- ▶ Reduce this set
- ▶ Remove the new resolvents including a clause from $\Delta_j \cup \Theta_j$

Θ_{i+1}

Remove from $\Delta_j \cup \Theta_j$ the clauses which include a clause from Δ_{j+1} .

When $\Delta_k = \emptyset$, stop the construction:

- ▶ $k - 1$ is then the maximum height of a proof
- ▶ Θ_k is the reduced set of the clauses deduced from Γ

Exemple 2.2.1

Soit $\Gamma = \{a + b + \bar{a}, a + b, a + b + c, a + \bar{b}, \bar{a} + b, \bar{a} + \bar{b}\}$

i	Δ_i	Θ_i	$\Delta_i \cup \Theta_i$	Résolvants de Δ_i et $\Delta_i \cup \Theta_i$

i	Δ_i	Θ_i	$\Delta_i \cup \Theta_i$	Résolvants de Δ_i et $\Delta_i \cup \Theta_i$
0	$a + b + \bar{a}, a + b,$ $a + b + c, a + \bar{b}$ $\bar{a} + b, \bar{a} + \bar{b}$	\emptyset	$a + b, a + \bar{b},$ $\bar{a} + b, \bar{a} + \bar{b}$	$a, b, b + \bar{b},$ $a + \bar{a}, \bar{b}, \bar{a}$

i	Δ_i	Θ_i	$\Delta_i \cup \Theta_i$	Résolvants de Δ_i et $\Delta_i \cup \Theta_i$
0	$a + b + \bar{a}, a + b$ $a + b + c, a + \bar{b}$ $\bar{a} + b, \bar{a} + \bar{b}$	\emptyset	$a + b, a + \bar{b},$ $\bar{a} + b, \bar{a} + \bar{b}$	$a, b, b + \bar{b},$ $a + \bar{a}, \bar{b}, \bar{a}$
1	a, b, \bar{b}, \bar{a}	\emptyset	a, b, \bar{b}, \bar{a}	\perp

i	Δ_i	Θ_i	$\Delta_i \cup \Theta_i$	Résolvants de Δ_i et $\Delta_i \cup \Theta_i$
0	$a + b + \bar{a}, a + b$ $a + b + c, a + \bar{b}$ $\bar{a} + b, \bar{a} + \bar{b}$	\emptyset	$a + b, a + \bar{b},$ $\bar{a} + b, \bar{a} + \bar{b}$	$a, b, b + \bar{b},$ $a + \bar{a}, \bar{b}, \bar{a}$

The proof we built

1	$a + b$	
2	$a + \bar{b}$	
3	$\bar{a} + b$	
4	$\bar{a} + \bar{b}$	
5	a	resolvent of 1 and 2
6	b	resolvent of 1 and 3
7	\bar{b}	resolvent of 2 and 4
8	\bar{a}	resolvent of 3 and 4
9	\perp	resolvent of 5 and 8

Example 2.2.2

$$\{a, c, \bar{a} + \bar{b}, \bar{c} + e\}$$

i	Δ_i	Θ_i	$\Delta_i \cup \Theta_i$	Rés. de Δ_i et $\Delta_i \cup \Theta_i$
0	$a, c, \bar{a} + \bar{b}, \bar{c} + e$	\emptyset	$a, c, \bar{a} + \bar{b}, \bar{c} + e$	\bar{b}, e
1	\bar{b}, e	a, c	\bar{b}, e, a, c	\emptyset
2	\emptyset	\bar{b}, e, a, c		

Rappel :

- ▶ $\Delta_{i+1} =$
 - ▶ Compute all the resolvents of Δ_i and $\Delta_i \cup \Theta_i$
 - ▶ Reduce this set
 - ▶ Remove the new resolvents which include a clause from $\Delta_i \cup \Theta_i$
- ▶ $\Theta_{i+1} =$
 - Remove from $\Delta_i \cup \Theta_i$ the clauses which include a clause of Δ_{i+1} .

Termination of the algorithm: idea

There are at most 2^n clauses deduced from Γ .

$\Delta_{i(i \geq 0)}$ contains only clauses deduced from Γ

Property 2.2.4

For all $i \leq k$, the sets Δ_i are mutually disjoint.
(by construction of Δ_i)

$\Delta_{i(i \geq 0)}$ are mutually disjoint

Hence there are at most $2^n + 1$ sets, therefore $k \leq 2^n + 1$

Result of the algorithm

When the algorithm terminates:

if $\perp \in \Theta_k$: Γ is **unsatisfiable**

if $\perp \notin \Theta_k$: Γ is **satisfiable**, but what does Θ_k represent?

- ▶ Θ_k = set of minimum deduction clauses.
- ▶ Γ and Θ_k are **equivalent**.

Property 2.2.5

For all $i < k$, the sets $\Delta_i \cup \Theta_i$ and $\Delta_{i+1} \cup \Theta_{i+1}$ are equivalent.

Hence :

$$\Gamma \equiv \Delta_0 \cup \emptyset = \Delta_0 \cup \Theta_0 \equiv \dots \equiv \Delta_k \cup \Theta_k = \emptyset \cup \Theta_k = \Theta_k$$

Today

- ▶ Resolution is a **correct** and **complete** deductive system: it **characterizes** all the unsatisfiable formulae.
- ▶ The **DPLL algorithm** uses ideas from resolution to:
 - ▶ find a **model**
 - ▶ or else, prove the **unsatisfiability** by an efficient search of the assignments.
- ▶ **Complete Strategy** is an **algorithm** for computing **every** clause deducible from an initial set

Next lecture

- ▶ Natural deduction

Homework: **Hypotheses** :

- ▶ (H1) : $p \Rightarrow \neg j \equiv \neg p \vee \neg j$
- ▶ (H2) : $\neg p \Rightarrow j \equiv p \vee j$
- ▶ (H3) : $j \Rightarrow m \equiv \neg j \vee m$
- ▶ ($\neg C$): $\neg m \wedge \neg p$ (two clauses)

Build the proof of $H1, H2, H3, \neg C \vdash \perp$ obtained by the DPLL algorithm (you may pick any variable for branching)